**Median of BST :-**

Easy Accuracy: 27.43% Submissions: 61K+ Points: 2

Given a Binary Search Tree of size N, find the Median of its Node values.

**Example 1:**

**Input:**

       6

     /   \

   3      8

 /  \    /  \

1    4  7  9

**Output:** 6

**Explanation:** Inorder of Given BST will be:

1, 3, 4, 6, 7, 8, 9. So, here median will 6.

**Example 2:**

**Input:**

       6

     /   \

   3      8

 /   \    /

1    4  7

**Output:** 5

**Explanation:**Inorder of Given BST will be:

1, 3, 4, 6, 7, 8. So, here median will

(4 + 6)/2 = 10/2 = 5.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **findMedian()** which takes the root of the Binary Search Tree as input and returns the Median of Node values in the given BST.  
Median of the BST is:

* If number of nodes are even: then median = (N/2 th node + (N/2)+1 th node)/2
* If number of nodes are odd : then median = (N+1)/2th node.

**Expected Time Complexity:**O(N).  
**Expected Auxiliary Space:**O(Height of the Tree).

**Constraints:**  
1<=N<=10000

**CODE :-**

//{ Driver Code Starts

#include <bits/stdc++.h>

using namespace std;

#define MAX\_HEIGHT 100000

// Tree Node

struct Node {

int data;

Node \*left;

Node \*right;

Node(int val) {

data = val;

left = right = NULL;

}

};

float findMedian(struct Node\* node);

// Function to Build Tree

Node\* buildTree(string str)

{

// Corner Case

if(str.length() == 0 || str[0] == 'N')

return NULL;

// Creating vector of strings from input

// string after spliting by space

vector<string> ip;

istringstream iss(str);

for(string str; iss >> str; )

ip.push\_back(str);

// Create the root of the tree

Node\* root = new Node(stoi(ip[0]));

// Push the root to the queue

queue<Node\*> queue;

queue.push(root);

// Starting from the second element

int i = 1;

while(!queue.empty() && i < ip.size()) {

// Get and remove the front of the queue

Node\* currNode = queue.front();

queue.pop();

// Get the current node's value from the string

string currVal = ip[i];

// If the left child is not null

if(currVal != "N") {

// Create the left child for the current node

currNode->left = new Node(stoi(currVal));

// Push it to the queue

queue.push(currNode->left);

}

// For the right child

i++;

if(i >= ip.size())

break;

currVal = ip[i];

// If the right child is not null

if(currVal != "N") {

// Create the right child for the current node

currNode->right = new Node(stoi(currVal));

// Push it to the queue

queue.push(currNode->right);

}

i++;

}

return root;

}

int main() {

int t;

string tc;

getline(cin, tc);

t=stoi(tc);

//cout << t << endl;

while(t--)

{

string s;

getline(cin, s);

Node\* root = buildTree(s);

// getline(cin, s);

cout << findMedian(root) << endl;

// cout<<"~"<<endl;

}

return 0;

}

// } Driver Code Ends

// your task is to complete the Function

// Function should return median of the BST

void number(Node \*root, int &count){

if(!root) return;

count++;

number(root->left, count);

number(root->right, count);

return;

}

void median(Node \*root, int &count, int sind, int eind, float &ans){

if(!root) return;

median(root->left, count, sind, eind, ans);

count++;

if(count==sind) ans = root->data;

if(count==eind) ans = (ans + (float)root->data) / 2;

median(root->right, count, sind, eind, ans);

return;

}

float findMedian(struct Node \*root){

int count=0, sind=0, eind=0;

float ans=0;

number(root, count);

//even number of nodes

if(count%2 == 0){

sind = count / 2;

eind = sind + 1;

}

//odd number of nodes

else if(count&1 == 1){

sind = (count/2) + 1;

eind = sind;

}

count = 0;

median(root, count, sind, eind, ans);

return ans;

}

**T.C :- O(N)**

**S.C :- O(Height) including recursion call stack**